Polynomial-Time Computation in the Pi-Calculus

A Mild Internship Report

Romit Roy Chowdhury, Patrick Baillot

Univ-Lille???

add(x, y) = x+y

```
add   0  y = y
add s(x) y = s (add x y)
```

(Ungodly mixture of Haskell and Coq)

exp(x) = 2^x

```
exp   0  = 1
exp s(x) = add(exp x, exp x)
```

add(x, y) = x+y

```
add   0  y = y
add s(x) y = s (add x y)
```

(Ungodly mixture of Haskell and Coq)

exp(x) = 2^x

```
exp   0  = 1
exp s(x) = (add . (id, id)) (exp x)
```

Primitive Recursion

- Basic Functions:
  1. Constant function: 0
  2. Successor function: $x \rightarrow x+1$
  3. Projection function: $(x_1, \ldots, x_n) \rightarrow x_k$

- Composition: $h(x) = f(g(x))$.

- PRIMITIVE RECURSION:

  $f(0, \vec{n}) = g(\vec{n})$

  $f(s(x), \vec{n}) = h(x, f(x, \vec{n}), \vec{n})$

<u>add(x, y) = x+y</u>

```
add   0  y = y
add s(x) y = s (add x y)
```

<u>mult(x, y) = x*y</u>

```
mult   0  y = 0
mult s(x) y = add (y (mult x y))
```

<u>exp(x, y) = y^x</u>

```
exp   0  y  = 1
exp s(x) y = mult (y (exp x y))
```

Where is the explosion?

exp(x) = 2^x

```
exp   0  = 1
exp s(x) = (add (exp x, exp x))
```

fac(x) = x!

```
fac   0  = 1
fac s(x) = mult (y (fac x))
```

add(x, y):

```
add   0  y = y
add s(x) y = s (add x y)
```

mult(x, y):

```
mult   0  y = 0
mult s(x) y = add (y (mult x y))
```

exp(x, y):

```
exp   0  y  = 1
exp s(x) y = mult (y (exp x y))
```

- Stephen Bellantoni, Stephen Cook (1992).

- SAFE RECURSION:

  f(x, y):

  $f(0, \vec{n}; \quad \vec{s}) = g(\vec{n}; \quad \vec{s})$

  $f(s(x), \vec{n}; \quad \vec{s}) = h(y, \vec{n}; \quad \vec{s}, f(x, \vec{n}, \vec{s}))$

- *The class of safe recursive functions is equal to the class of polynomial-time functions.*

PI CALCULUS.

$\mid \quad a(x)\ b(y) \quad \mid \quad \bar{b}\langle t\rangle \quad \mid$

$\mid \quad \bar{a}\langle s\rangle \quad \mid$

$P, Q :=$	0	null			
$\mid$	$\bar{a}\langle x\rangle$	output	$\mid$	$\nu a.P$	new
$\mid$	$a(x).P$	input	$\mid$	$!P$	replication
$\mid$	$P \mid Q$	parallel	$\mid$	$P + Q$	choice

Communication:

$a(x).P \quad \mid \quad \bar{a}\langle b\rangle \quad \longrightarrow \quad P$(where x is replaced by b) $\quad \mid \quad 0$

$\longrightarrow \quad P[{}^{b}/_{x}] \quad \mid \quad \bar{a}\langle b\rangle$

- val = 1,00,000

  $\overline{a}\langle m\rangle \quad | \quad a(x)\,\overline{x}\langle \text{val}\rangle \quad | \quad \overline{a}\langle n\rangle$

  $\longrightarrow \qquad \overline{m}\langle \text{val}\rangle \quad | \quad \overline{a}\langle n\rangle$

  or, $\quad \overline{a}\langle m\rangle \quad | \quad \overline{n}\langle v\rangle$

- m = 20, n = 13

  $\nu w.(\overline{a}\langle w\rangle \,|\, \overline{w}\langle m\rangle) \quad | \quad a(x).x(p).\overline{x}\langle \text{val}\rangle \quad | \quad \nu u.(\overline{a}\langle u\rangle \,|\, \overline{u}\langle n\rangle)$

  $\longrightarrow \quad \overline{w}\langle \text{val}\rangle \quad | \quad (\overline{a}\langle u\rangle \,|\, \overline{u}\langle n\rangle)$

Replication

- Extra: NUMBERS!

  $n ::= \quad 0 \quad | \quad \mathrm{succ}(n)$

  Number $n == 0$?

- $!s(x).\bar{s}\langle x+1\rangle \quad | \quad (\bar{s}\langle 5\rangle \mid s(x).P)$ bad

- r: return channel

  $!s(x,r).\bar{r}\langle x+1\rangle \quad | \quad \nu r.\,(\bar{s}\langle 5,r\rangle \mid r(x).P)$ magic!

- $!P \quad \equiv \quad (!P \quad | \quad P)$

$N := \{a, b, c, \dots\}$

$|$ $|$ $|$

$|$ $|$ $|$

(BOARD)

Functions in pi-calculus

$$
\begin{aligned}
&!\mathrm{add}(x, y, r).\,\mathrm{match}\\
&[x = 0]\,\bar{r}\langle y\rangle \,+\\
&[x = s(x')]\,\nu c\,(\overline{\mathrm{add}}\langle x', y, c\rangle \quad | \quad c(v)\,\bar{r}\langle s(v)\rangle)
\end{aligned}
$$

$$
\begin{aligned}
&!\mathrm{mult}(x, y, r).\,\mathrm{match}\\
&[x = 0]\,\bar{r}\langle 0\rangle \,+\\
&[x = s(x')]\,\nu c, \nu d\,(\overline{\mathrm{mult}}\langle x', y, c\rangle \quad | \quad c(t)\,\overline{\mathrm{add}}\langle y, t, d\rangle \quad | \quad d(v)\,\bar{r}\langle v\rangle)
\end{aligned}
$$

Typing Systems

- Why do we want to use types?

- Many processes can have meaningless computations.

- For example we don't want to do:

  $x = ''\text{Imperative programming}''; \quad y = x + 1$

- Similarly,

  $\nu w.(\overline{a}\langle w\rangle \quad | \quad \overline{w}\langle 20\rangle) \quad | \quad (a(x).\overline{a}\langle x+1\rangle)$

- Type mismatch! Breaking expectation.

$$N := \quad \mathbb{N} \quad | \quad \mathrm{ch}(N_1, \ldots, N_k)$$

- Prevents numbers being used as channel names.

- Use expectation to enforce stronger properties.

  Suppose your process takes a $\mathbb{N}$ as input but only expects naturals of a certain size.

$$N := \quad \mathbb{N}[j] \quad | \quad \mathrm{ch}(N_1, \ldots, N_k)$$

BG Typing System

- Patrick Baillot, Alexei Ghyselen (2020).

- Sized Types for Pi-Calculus Processes:

  $$N := \mathbb{N}[e(I), e(J)] \quad | \quad \mathrm{ch}(N_1, \ldots, N_k) \quad | \quad \mathrm{serv}(N_1, \ldots, N_k)$$

- *If you can sucessfully give a BG-type to the names in a process using the typing rules, then a bound on the size of $\mathbb{N}$ passed as well as on the total number of reduction steps is guaranteed.*

- With polynomial expressions for e, we can create polytime processes.

- SAFE RECURSION!
- Demangeon, Yoshida (2021).
- Typing system which separates normal values and safe values!
- $N ::= \quad \mathbb{N}_{\text{normal}} \quad | \quad \mathbb{N}_{\text{safe}} \quad | \quad \mathrm{ch}(N_1, \ldots, N_k)$
- *Any process which can be typed in the DY system is guaranteed to compute only polynomial-time functions and have polynomial reduction steps (roughly).*

$!\mathrm{add}(x, y, r).\,\mathrm{match}$

$[x = 0]\,\bar{r}\langle y \rangle\; +$

$[x = s(x')]\,\nu c\,(\overline{\mathrm{add}}\langle x', y, c \rangle \;\;|\;\; c(v)\,\bar{r}\langle s(v) \rangle)$

$x\colon \mathbb{N}_{\mathrm{normal}}$

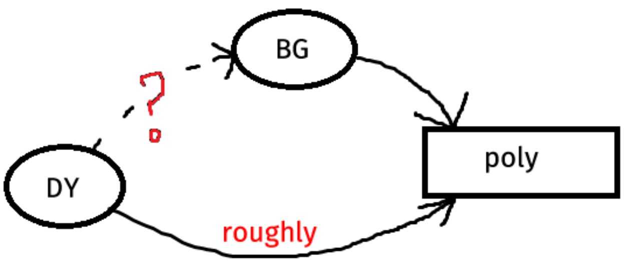
$y\colon \mathbb{N}_{\mathrm{safe}}$

$r\colon \mathrm{ch}(\mathbb{N}_{\mathrm{safe}})$

$c\colon \mathrm{ch}(\mathbb{N}_{\mathrm{safe}})$

$\mathrm{add}\colon (\mathbb{N}_{\mathrm{normal}}, \mathbb{N}_{\mathrm{safe}}, \mathrm{ch}(\mathbb{N}_{\mathrm{safe}}))$

The plan

- Idea: Take any process that can be DY-typed, and give it a BG-typing.
- 

- General procedure that would work for any process, giving a proof of the DY to BG translation.
- Some problems: BG is unable to handle some types of processes.

  Characterize classes?

What about PSPACE?

- What about PSPACE?

Thank You!